Rapid application prototyping for hardware modular spiking neural network architectures

Neural Computing and Applications manuscript No.
(will be inserted by the editor)

Rapid Application Prototyping for Hardware Modular Spiking Neural Network Architectures

Sandeep Pande · Fearghal Morgan · Finn Krewer · Jim Harkin · Liam McDaid · Brian McGinley

Received: date / Accepted: date

Abstract Spiking Neural Networks (SNNs) are well suited for functions such as data/pattern classification, estimation, prediction, signal processing and robotic control applications. Whereas, the real-world embedded applications are often multi-functional with orthogonal functional requirements. The EMBRACE hardware modular SNN architecture has been previously reported as an embedded computing platform for complex real-world applications. The EMBRACE architecture employs Genetic Algorithm (GA) for training the SNN which offers faster prototyping of SNN applications, but exhibits a number of limitations including poor scalability and search space explosions for the evolution of large scale, complex, real-world applications. This paper investigates the limitations of evolving real-world embedded applications with orthogonal functional goals on hardware SNN using GA-based training.

This paper presents a fast and efficient application prototyping technique using the EMBRACE hardware modular SNN architecture and the GA-based evolution platform. Modular design and evolution of a robotic navigational controller application decomposed into obstacle avoidance controller and speed and direction manager application subtasks is presented. The proposed modular evolution technique successfully integrates the orthogonal functionalities of the application and helps to overcome contradicting application scenarios gracefully. Results illustrate that the modular evolution of the application reduces the SNN configuration search space and complexity for the GA-based SNN evolution, offering rapid and successful prototyping of complex applications on the hardware SNN platform. The paper presents validation results of the evolved robotic application implemented on the EMBRACE architecture prototyped on Xilinx Virtex-6 FPGA interacting with the player-stage robotics simulator.

Sandeep Pande, Fearghal Morgan, Brian McGinley, Finn Krewer
Bio-Inspired Electronics and Reconfigurable Computing,
National University of Ireland, Galway, Ireland.
E-mail: sandeep.pande@ieee.org

Jim Harkin, Liam McDaid
Intelligent Systems Research Centre,
University of Ulster, Derry, Northern Ireland, UK.

1 Introduction

Classical computing paradigms are suitable for applications comprising a series of well-defined calculations. Traditional computing techniques have a number of limitations when applied to real-world embedded applications. These applications are often characterised by the contradicting functional requirements in unexplored data and application scenarios. Hence the perfect solutions (that offer required behaviour in all the possible application scenarios) for these applications are difficult due to the inherent non-linearity.

Biologically inspired artificial neural network computing techniques mimic the key functions of the human brain and have the potential to offer smart and adaptive solutions for complex real-world problems [1]. The organic nervous system includes a dense and complex interconnection of neurons and synapses, where each neuron connects to thousands of other neurons through synaptic connections. Computing systems based on Spiking Neural Networks (SNNs) emulate real biological neural networks, conveying information through the communication of short transient pulses (*spikes*) via synaptic connections between neurons [2] [3]. Embedded systems based on hardware SNNs offer elegant solutions as low-power and scalable embedded computing elements, characterised with rich non-linear dynamics. Hardware SNNs are suited to real-world applications including data/pattern classification, estimation, prediction, dynamic/non-linear control and signal processing [4] [5].

The authors have investigated and proposed EMBRACE[1] as an embedded computing architecture for the implementation of large scale SNNs [6] [7] [8]. The EMBRACE architecture comprises Modular Neural Tiles (MNTs) interconnected using a hierarchical Network on Chip (NoC) communication infrastructure [9] [10]. EMBRACE architecture prototype uses a Genetic Algorithm (GA)-based SNN evolution and configuration platform that configures the SNN synaptic weights and neuron threshold potentials and evolves the desired functionality by searching the correct SNN configuration (synaptic weights and threshold potential). The GA-based search executing on the host computer interfaces only with the overall input and output of the SNN. This technique offers a simple method of prototyping applications on hardware SNN platforms. The authors have successfully applied the EMBRACE hardware SNN system to a number of benchmark data classification and control applications such as XOR data classifier, inverted pendulum controller, Wisconsin breast cancer dataset classifier and robotic controllers [7] [8] [11].

Real-world embedded applications are often multi-functional with orthogonal functional requirements. Perfect solutions for these applications are difficult due to the absence of an exact or deterministic computational algorithm. The inherent fuzzy nature and contradicting functional requirements of these applications poses a serious challenge for their implementation on hardware SNN architectures. The GA-based SNN training has been effective for evolving small uni-functional applications on hardware SNN platforms. This technique offers a simple and rapid prototyping method for the validation of hardware SNN platforms [7] [8] [11]. However, the GA-based SNN evolution technique exhibits a number of limitations including poor scalability and search space explosion for the evolution of complex SNN applications. The paper illustrates the limitations of a GA-based SNN evolution in solving increasingly complex applications with orthogonal functional goals such

[1] **EM**ulating **B**iologically-inspi**R**ed **A**r**C**hitectures in hardwar**E**

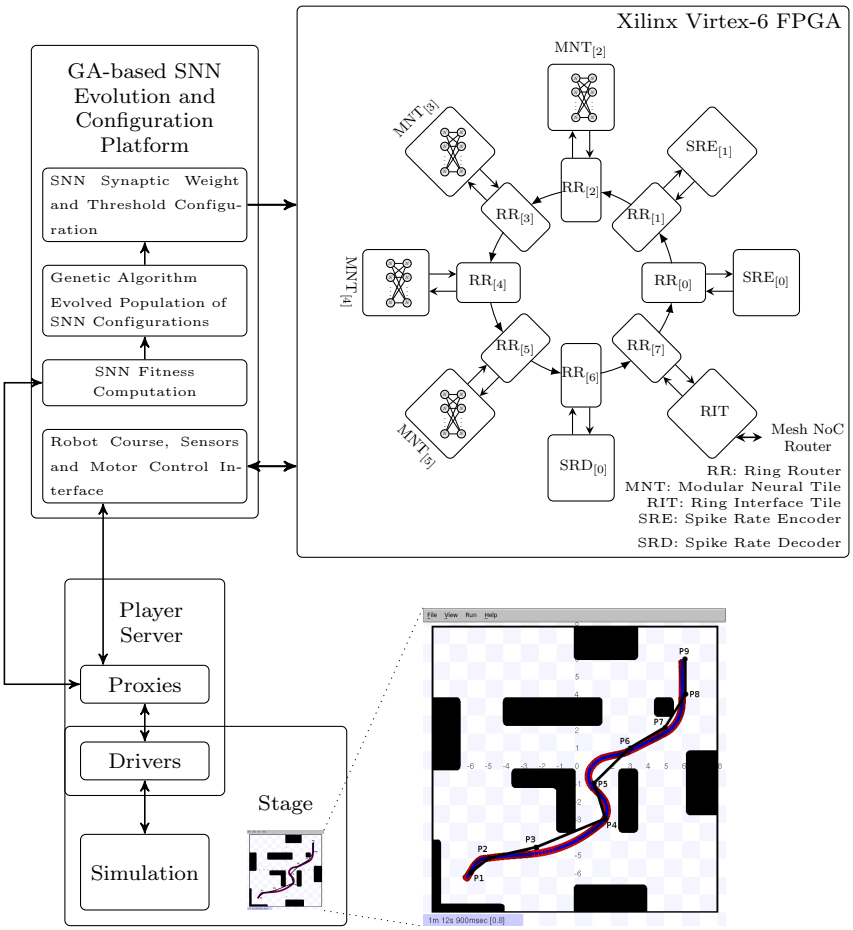

Fig. 1: EMBRACE Hardware SNN Robotic Controller Application Evolution System

as a robotic controller application with contradicting behavioural requirements in certain scenarios.

The limitations of GA-based SNN evolution can be mitigated using the classic *divide and conquer* technique, by defining the overall behaviour as a number of less complex orthogonal functions. The *Subsumption Architecture* is a layered application organisation which is based on the divide and conquer technique. The subsumption architecture suggests partitioning the high level behavioral robotic applications into layers of control modules, where higher level layers subsume the roles of lower layer functions [12]. A Modular Neural Network (MNN) architecture comprises interconnected neural computation modules, and hence lends itself to subsumption-style function implementation.

This paper presents a rapid application prototyping technique for the EMBRACE hardware SNN architecture using the subsumption architecture based modular application partitioning technique. This mitigates the problems associated

with the GA-based SNN evolution for complex real-world embedded applications and offers a rapid yet simple application prototyping technique for hardware SNN architectures. The technique has been applied to a robotic navigational controller application which directs the robot on a pre-determined route in a two-dimensional environment containing obstacles. The multifunctional robotic navigational controller application has been decomposed into the Obstacle Avoidance Controller (OAC) and the Speed and Direction Manager (SDM) application subtasks. The OAC subtask uses sensory inputs to steer the robot avoiding obstacles while the SDM subtask manoeuvres the robot towards the target location. The Integration (INT) subtask combines the output from both the OAC and SDM tasks to provide the overall control to the robot in its environment.

Figure 1 illustrates the robotic controller application evolution setup comprising the EMBRACE hardware SNN architecture FPGA prototype [8] [9] interfaced with the player-stage robotics simulator [13] and GA-based hardware SNN evolution platform executing on the host computer. The individual application subtasks (OAC and SDM) and integration subtask (INT) are implemented on separate hardware SNN modules and evolved separately. The overall robotic behaviour is elaborated by combining the OAC and SDM output in the INT subtask.

This paper presents the robotic navigational controller modular application design including SNN application topology, GA fitness criteria and application evolution. Results illustrate that the modular evolution of the application reduces the SNN configuration search space and complexity (for the GA-based SNN evolution), offering rapid prototyping of complex applications on the hardware SNN platform. The modular application design approach offers simplified SNN training and faster application evolution compared to evolution of the complete application.

The structure of the paper is as follows: Section 2 reviews the MNN computing paradigm, hardware neural network execution architectures, modular application design and SNN training challenges. Section 3 presents the EMBRACE hardware SNN architecture (comprising MNTs interconnected using ring topology NoC) and FPGA prototype implementation. Section 4 presents the modular robotic navigational controller application design and evolution. Section 5 concludes the paper.

2 Related Work

This section reviews the Modular Neural Network (MNN) computing paradigm and discusses the MNN application design and execution architectures. SNN training techniques are reviewed to highlight their suitability for rapid application prototyping on hardware SNN architectures.

2.1 Modular Neural Network Computing Paradigm

The biological brain is considered to be composed of several anatomically and functionally discrete areas which process various sensory and motor tasks, and different aspects of information [14] [15] [16]. This modular organisation observed in the brain has been the inspiration behind the MNN computing paradigm [17]. The MNN computing paradigm is primarily based on the *divide and conquer*

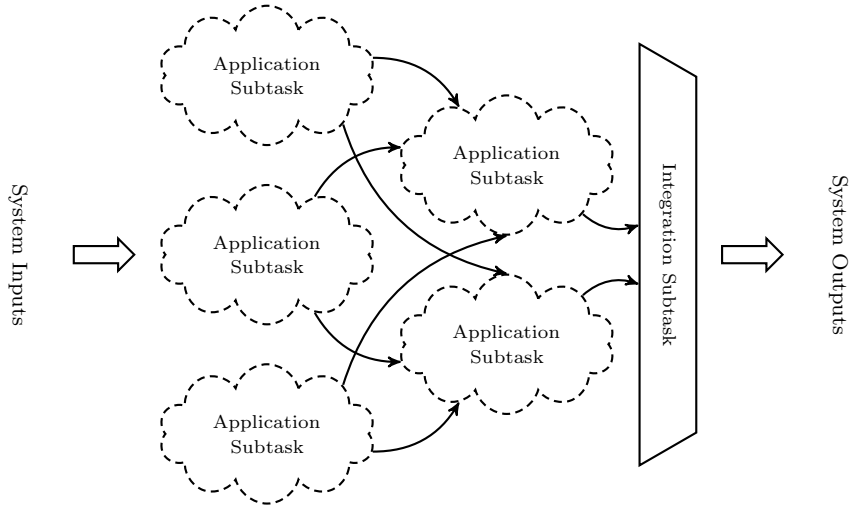

Fig. 2: Modular Neural Network Application Organisation

strategy. The MNN design strategy proposes partitioning of application into a number of subtasks executing distinctly on neural modules [17] [18] [19]. Figure 2 illustrates MNN application organisation, where *task decomposition* of the high level application leads to smaller, less complex and manageable subtasks which are solved by individual and distinct neural modules. The intermediate outputs from these neural modules are combined to solve the high level task or the complete application [1] [20].

2.1.1 MNN Application Design

Application design for MNN systems primarily involves *Task Decomposition* to find the correct set of application subtasks in order to achieve the overall application behaviour. The task decomposition algorithms analyse the overall application based on output vector partitioning and class relationships to obtain orthogonally functional subtasks [21] [22]. Neuro-evolutionary and co-evolutionary methods have also been proposed for the MNN design strategy [23] [24] [25]. Subtasks or subnetworks obtained after task decomposition are executed in neural modules within an MNN computing architecture.[2]

The MNN approach offers reduced training time, improved operation accuracy, structured implementation, functional partition, functional mapping and re-mapping, potentially competitive and co-operative mode of operation and fault tolerance [18]. A survey of MNN application designs is presented in [18] [19]. The *Subsumption Architecture* is a widely influential computing paradigm for reactive, behaviour-based autonomous robotic control applications. The subsumption

[2] The task decomposition for the robotic navigational controller modular application design presented in this paper has been done manually based on input vector partitioning and analysing the orthogonality of the functional requirements of the application.

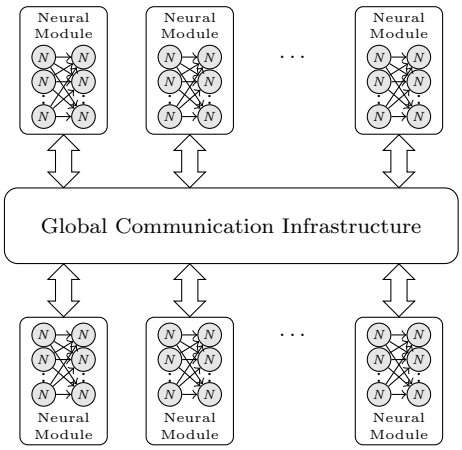

Fig. 3: Modular Neural Network Execution Architecture

architecture has been developed based on the MNN design concepts [12]. The subsumption architecture is a layered application organisation, used to partition high level behavioral robotic applications into layers of control modules, where higher level layers subsume the roles of lower layer functions. The MNN system architecture comprises interconnected neural computation modules, and hence lends itself to the subsumption-style function implementation.

2.1.2 MNN Execution Architectures

Task decomposition of an application leads to two types of subtasks; namely the *Application Subtasks* (discrete functional subtasks) and the *Integration Subtasks*. The application subtasks operate on individual and distinct system inputs to provide intermediate outputs. The integration subtasks integrate the intermediate outputs from the application subtasks to generate the overall system output. Both the application and integration subtasks have similar input-output interface and computational requirements and can be realised using a densely connected neural network.

Figure 3 illustrates a typical MNN execution architecture comprising individual neural modules interconnected by a global communication infrastructure. Each neural module is a fully connected feedforward topology neural network that includes a group of neurons interconnected using an internal communication infrastructure and has multiple inputs and/or multiple outputs. The global (or inter-module) communication infrastructure provides connectivity between the individual neural modules [1] [20].

2.2 Spiking Neural Network Training Challenges

SNN training process involves iterative adjustment of the SNN configuration (synaptic weights, threshold potential and/or network topology) to achieve the

desired application behaviour. This section presents a brief review of the SNN training algorithms that can be employed for training practical applications on hardware SNN architectures.

Neural network training algorithms can be broadly classified as:

- **Unsupervised Training Algorithms:** These training algorithms rely on tuning the neural network to statistical regularities of the input data, such that the neural network produces the correct outputs based on these input data patterns. Hebbian learning suggests the synaptic weight adjustment based on the firing sequence of the connected neurons and results in emergence of new functions, such as pattern recognition and associative memories [26] [27]. The probability of a neuron firing based on the pre-synaptic spike timing is termed as Spike Timing Dependent Plasticity (STDP). STDP based unsupervised SNN training techniques have been effective for data/pattern classification, pattern recognition and associative memories [28] [29] [30] [31] [32] [33].
- **Supervised Training Algorithms:** These algorithms are primarily characterised by the iterative adjustment of the neural network configuration based on feedback from a training data set. Back-Propagation and Spike-Prop SNN training algorithms are the most popular neural network supervised training algorithms [34] [35]. Supervised Hebbian Learning (SHL) offers a biologically realistic implementation of Hebbian learning rules [36] [37].
- **Reinforcement Training Algorithms:** These algorithms tune the neural network configurations to achieve a predetermined goal, while interacting with the environment. The technique relies on rewarding the performance improving configurations and avoiding the performance deteriorating configurations [38] [39] [40] [41] [42] [43].
- **Evolutionary Training Algorithms:** Evolutionary methods have been successfully applied for SNN training and evolution of robotic applications [44] [45]. GA-based SNN evolution is a reward based SNN training method that maintains a population of SNN configurations and uses nature inspired evolutionary techniques (comprising selection, crossover and mutation) to find a correct set of SNN configuration by evaluating the desired behaviour or 'fitness' of the individual SNN configurations. The technique only uses the 'fitness' value of the individual SNN configurations to compute the next set of SNN configurations and does not require the individual neuron firing time details (as compared to SHL/STDP-based techniques). Hence, the technique does not impose special architectural constraints or require additional training specific elements (such as relative timings between input-output spikes) in the neural computation components and results in a compact architecture [46]. The GA-based SNN evolution executing on host machine offers an easy and fast way to prototype applications on hardware SNNs.
  However, the GA-based SNN evolution does not scale well for large multi-functional applications having complex fitness landscape. The search space for the binary coded, integer valued, n bit SNN configuration is 2^n. Increase in the SNN size (number of neurons and synapses) increases the GA search space and complexity by the factor of $O(2^n)$ (also termed as 'search space explosion'). The technique also requires large amount of memory for storage of SNN configurations and results in slower operation due to iterative nature

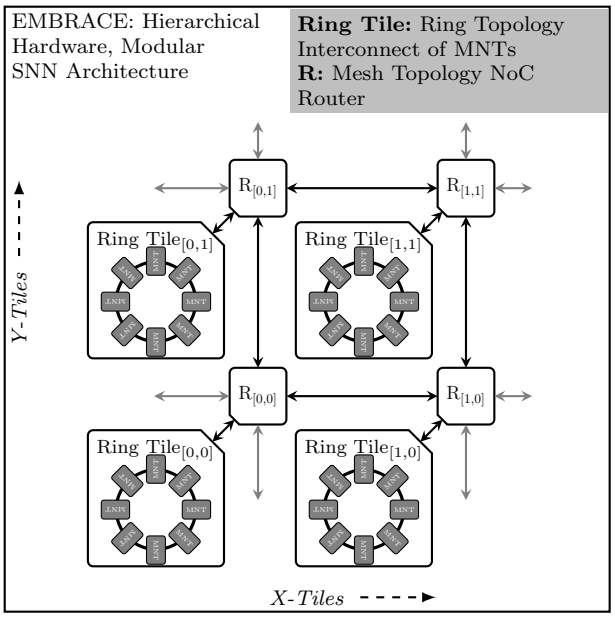

Fig. 4: EMBRACE Hardware Modular SNN Architecture with Hierarchical NoC Comprising Ring Topology Interconnects within Mesh Topology Routers

of SNN configuration evaluation. These shortcomings limit its use for evolving large real-world complex SNN applications and also in embedded systems.

The modular application prototyping presented in this paper reduces the search space and complexity for the GA-based SNN evolution offering fast evolution of complex, multifunctional applications. The proposed technique mitigates the weaknesses of the GA-based SNN evolution and helps rapid prototyping of real-world complex embedded applications on the EMBRACE hardware SNN platform.

3 EMBRACE Hardware Modular Spiking Neural Network Execution Architecture

This section presents the previously reported EMBRACE hardware modular SNN architecture as an embedded computing platform for real-world applications. The EMBRACE architecture depicted in figure 4 comprises Modular Neural Tiles (MNTs) interconnected using a scalable and hierarchical NoC [8] [9] [10].

3.1 Modular Neural Tile Architecture

The EMBRACE MNT architecture is a two-layered 16:16 fully connected feed-forward SNN structure as illustrated in figure 5) [8] [10]. The input layer ($N[0,n]$) and output layer ($N[1,n]$) of the MNT comprises 16 Leaky-Integrate-and-Fire (LIF) neurons. Each LIF neuron maintains a *membrane potential*, which is a function of

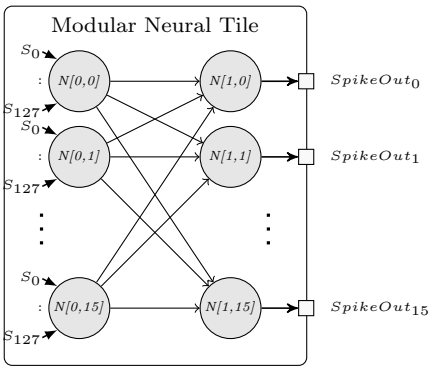

Fig. 5: Two layered 16:16 Fully Connected SNN Structure as Modular Neural Tile

incoming spikes, associated synaptic weights, current membrane potential, and the membrane potential *leakage coefficient* [2] [3]. A neuron *fires* (emits a spike to all connected synapses/neurons), when its membrane potential exceeds the neuron's firing threshold value. Each input layer neuron has 128 synapses corresponding to each output layer neuron within the ring interconnect. The MNT has 16 spike outputs ($SpikeOut_n$) each corresponding to the 16 output layer neurons.

The EMBRACE MNT architecture prototype on Xilinx Virtex-6 FPGA and micro-architectural details of the digital neuron model have been presented in [8]. The EMBRACE MNT FPGA prototype has been successfully applied to benchmark SNN application tasks representing classification and non-linear control functions [8].

3.2 EMBRACE Hierarchical NoC Architecture

The NoC-based synaptic connectivity approach provides flexible inter-neuron communication channels, scalable interconnect and connection reconfigurability [11] [47]. The EMBRACE spike communication infrastructure is a hierarchical, bi-level NoC architecture. The MNTs are grouped and interconnected through a synchronous ring topology interconnect. The MNT groups (organised as ring tiles) are interconnected via the upper level mesh topology network of routers. The following subsections describe these interconnects and the spike flow between MNTs.

3.2.1 Ring Interconnect Architecture

Modular SNNs exhibit a high density of synaptic connections within localised groups. The ring topology interconnect offers fixed spike transfer latency for inter-MNT spike communication within the ring [9]. The ring interconnect comprises routers connected in a unidirectional ring topology. The MNTs interface with ring routers for spike communication. Each router buffers the spike events generated by the attached MNT and encodes them in spike packets. Generated spike packets are processed and forwarded in a single clock cycle. Full rotation of the spike packet on the ring ensures broadcast packet flow control.

Fixed latency flow control of the ring topology interconnect uses timestamping of the input spikes at the source router, and broadcasting of spike packets to all routers within the ring. Each destination router sorts and buffers spike events based on the received spike packet timestamp value. Buffered spike events are converted to output spikes at the precise clock cycle (corresponding to the source timestamp) determined by the timeslot counter. The ring NoC interconnect offers fixed latency spike communication which eliminates information distortion in SNNs and results in stable and reliable application behaviour [8]. The ring NoC interconnect is particularly suitable for the 'spike time coded' SNNs and STDP training algorithms.

3.2.2 Mesh NoC Architecture

Each ring contains seven MNTs and a Ring Interface Tile (RIT) (instantiated as tile number $n = 7$ figure 1). The RIT acts as the bridge between the ring interconnect and mesh NoC router, facilitating the synaptic connectivity between MNTs within the ring and the rest of the chip [9]. An incoming spike on an input synapse of the RTI is converted to a spike packet by the packet encoder and forwarded to the attached NoC router. The NoC architecture supports unicast packet flow control, where each spike packet contains destination synapse information for a single spike and is routed independently. Different SNN topologies are created by configuring traffic connections between SNN elements (neurons, spike rate encoders and decoders).

4 Modular Application Design

Real-world applications are multi-functional with many behavioural aspects described as exceptions. Unexplored conditions add another dimension of complexity in clearly describing its behaviour. Hence an exact algorithmic solution is hard to derive, but a reasonably good solution can be built using a number of application modules each solving a distinct orthogonal functional requirement. This section highlights modular application design for hardware SNN architectures based on the subsumption architecture design principles. The section presents the modular robotic navigational controller application design including task decomposition, subtask design and evolution on EMBRACE-FPGA. The proposed technique is best suited for SNN application prototyping on hardware SNN platforms and the hardware SNN architectural validation. Although, the proposed problem consists of a simple robotic navigational controller with two functional aspects, it can be extended for large multi-functional applications.

The subsumption architecture proposes decomposition of the overall application in small sized subtasks arranged in a layered fashion, where each subtask addresses a specific application functionality [12]. Based on the subtask functionality, individual subtasks can be supplied with all or a subset of the system inputs. Subtask outputs are combined into higher level tasks or integration subtasks. The final system output is always produced by the integration subtasks based on various intermediate output patterns.

In summary, an application can be primarily decomposed based on:

- sensory inputs

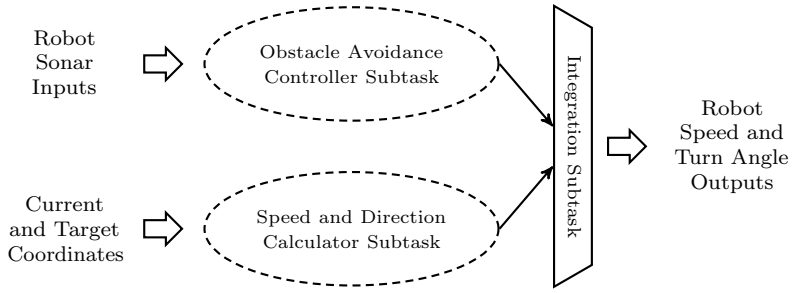

Fig. 6: Robotic Navigational Controller Application Organisation

- application goals or functions
- functional redundancy (multiple application modules assigned for the same functionality to increase robustness)
- structured application layers (functional levels during the integration phase)
- application extensibility

The task decomposition presented in this paper is done based on the input vector partitioning and deriving simple orthogonal functional requirements of the application.

4.1 Robotic Navigational Controller Application Design

The primary goal of the proposed robotic navigational controller application is to direct the robot on a pre-determined route in a two-dimensional simulated environment containing obstacles. The pre-determined route (selected by the designer) is marked with fixed points (or markers)[3] and the two-dimensional simulated world contains obstacles that must be avoided by the robot (depicted in figure 1). The two primary sub-functions derived from the main application are

- obstacle avoidance in the two-dimensional environment and
- speed and turn angle calculation to reach the next marker.

The obstacle avoidance function requires information about the robot surrounding, obtained through the sonar proximity sensors on the robot. The speed and turn angle calculation function requires information about the current robot position and orientation, and the next marker coordinate. Based on these functional aspects and sensory input requirements, the robotic navigational controller can be partitioned into **Obstacle Avoidance Controller** (OAC) and **Speed and Direction Manager** (SDM) subtasks. Figure 6 illustrates the robotic navigational controller application organisation. Both the OAC and SDM subtasks produce speed and turn angle output suitable for manoeuvring the robot for obstacle avoidance and advancing towards target respectively. These subtask outputs can be integrated to produce actuation signals (speed and turn angle) for the robot.

[3] The markers for the proposed robotic navigational controller application are chosen by the designer such that the robot can progress towards the destination by following the markers in sequence. The markers can also be chosen by applying meta-heuristic algorithms to the classical *Travelling Salesman Problem* which is currently out of scope of this paper.

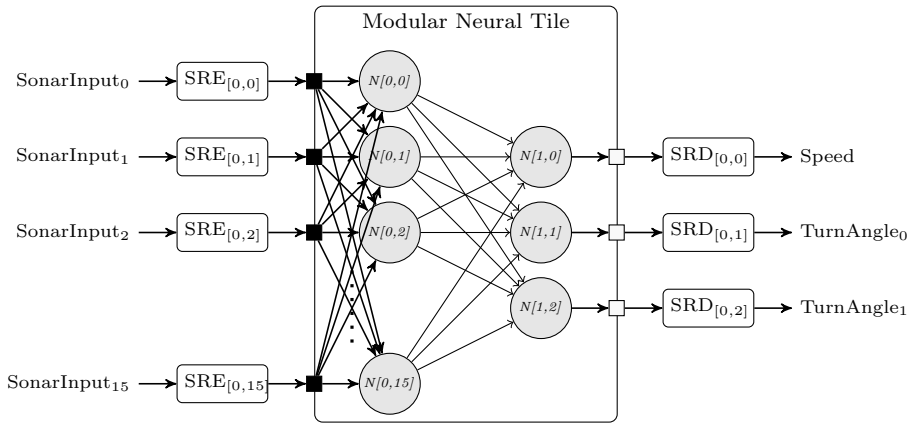

Fig. 7: Obstacle Avoidance Controller Subtask SNN Topology

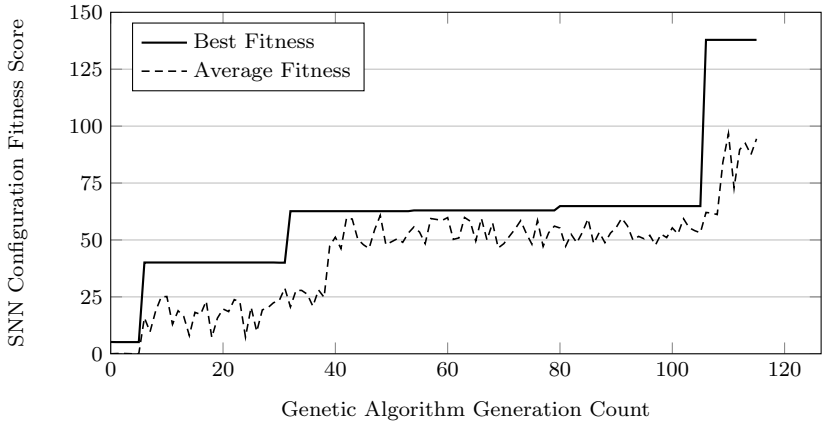

Fig. 8: Obstacle Avoidance Controller Subtask Evolution (Fitness Evaluation Constants: $\alpha = 0.25$ and $\beta = 0.125$)

4.2 Obstacle Avoidance Controller Subtask Design

The Obstacle Avoidance Controller (OAC) controls the speed and turn angle of the robot to avoid collisions with the obstacles inside the two-dimensional simulated robotic environment (figure 1). The simulated robot is equipped with 16 sonar proximity sensors which act as input to the OAC subtask SNN. The OAC subtask SNN topology illustrated in figure 7 is mapped to $\text{MNT}_{[2]}$ in figure 1. Robot sonar values are converted to spike rates by the host application and are passed to the spike rate encoders ($\text{SRE}_{[0]}$), which generate spikes into the MNT. Spikes from three MNT outputs (corresponding to speed and differential turnangle) are monitored by the spike rate decoders ($\text{SRD}_{[0]}$) and are converted to analogue values as robot speed and turning angle inputs to the simulator (where, Turning angle =

TurnAngle_0 - TurnAngle_1). The use of differential outputs for the calculation of turnangle eliminates the directional bias by individual outputs.

The fitness criteria for the OAC application is defined as robot travelling a finite distance and avoiding obstacles for $t \geq 120$ seconds. The fitness of the SNN configuration is calculated as:

$$F_{OAC} = \alpha T + \beta D \tag{1}$$

Where:

F_{OAC} = Fitness of the individual (for OAC subtask)
T = Robot travel time (in *seconds*)
D = Robot travel distance (in *cms*)

Given:

Number of collisions = 0
SNN outputs are within the operating range ($0 \leq Speed \leq 5$ and $-3.14 \leq TurnAngle \leq 3.14$)
Robot does not spin indefinitely
The fitness evaluation constants α and β are chosen to scale the outputs to match the time and distance measurement units.

Evaluation of the individual configuration has been accomplished with the robot roaming within the simulated environment for $120s \leq t \leq 300s$. On timeout ($t = 300s$), or if the robot has collided with an obstacle, the GA-based evolution and configuration platform processes the recorded robot behaviour and assigns a fitness score to the individual SNN configuration. Fitness scores are used by the GA to determine the probability of an individual SNN configuration progressing to the subsequent evolved generation of individual SNN configurations. Figure 8 illustrates the average and best fitness of the evolved OAC application subtask on the MNT. The average and best SNN configuration fitness score improves as the GA evolves leading to the successful obstacle avoidance by the robot for the full measurement time ($t = 300s$).

4.3 Speed and Direction Manager Subtask Design

The Speed and Direction Manager (SDM) subtask computes required speed and turn angle to advance the robot to the next marker. The robot navigates by presenting a series of markers to the SDM subtask. The SDM subtask takes the current robot coordinates and orientation (from positioning sensors on the robot) and the target marker coordinates as input and generates speed, and the turn angle. The ideal speed and turn angle are calculated using trigonometric equations (2) (3) (4) (5). The SDM application subtask is evolved to minimise the error function E_{SDM} elaborated in equation (6).

$$D = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \tag{2}$$

$$S_C = \gamma D \tag{3}$$

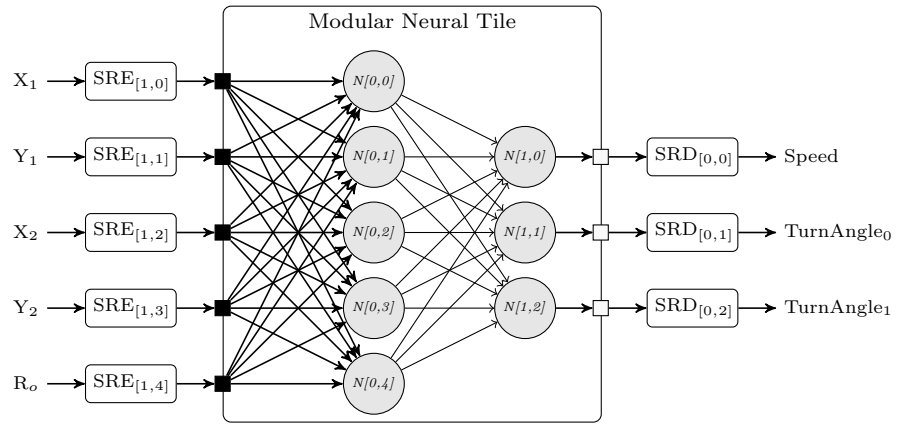

Fig. 9: Speed and Direction Manager Subtask SNN Topology

$$\theta = tan^{-1}\left(\frac{Y_2 - Y_1}{X_2 - X_1}\right) \tag{4}$$

$$T_{a(C)} = \theta - R_o \tag{5}$$

$$E_{SDM} = \mid S_C - S_M \mid + \mid T_{a(C)} - T_{a(M)} \mid \tag{6}$$

Where:

X_1, Y_1 = Current Coordinate
X_2, Y_2 = Target marker Coordinate
D = Distance between (X_1, Y_1) and (X_2, Y_2)
γ = 0.475
S_C = Calculated Robot speed
θ = Angle for (X_1, Y_1) to (X_2, Y_2)
R_o = Current robot orientation
$T_{a(C)}$ = Calculated Robot Turn Angle
$T_{a(M)}$ = Measured Robot Turn Angle
S_M = Measured Robot speed
E_{SDM} = SDM Error Function

The SDM subtask SNN topology illustrated in figure 9 is mapped to $\text{MNT}_{[3]}$ in figure 1. The current and target marker coordinates are converted to the spike rates by the host application and are passed to spike rate encoders ($\text{SRE}_{[1]}$), which generate spikes into the MNT. Spikes from three MNT outputs are monitored by the spike rate decoders ($\text{SRD}_{[0]}$) and are converted to analogue values as robot speed and turning angle (Turning angle = TurnAngle_0 - TurnAngle_1).

The evolution process uses a number of combinations of current and target marker coordinates to ensure minimum calculation error. Figure 10 illustrates average and best error values for the evolution of the SDM application subtask on the MNT. The error value represents the deviation of the calculated speed and

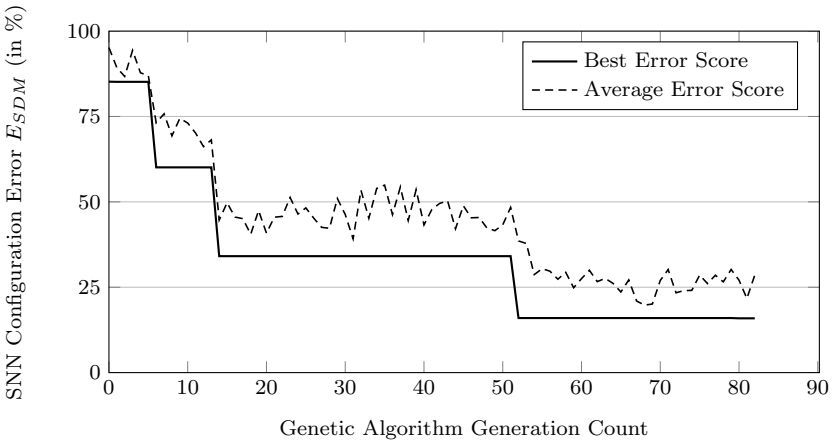

Fig. 10: Speed and Direction Manager Subtask Evolution (Fitness Evaluation Constants: $\gamma = 0.475$)
(Note: The measured error is capped at 100%.)

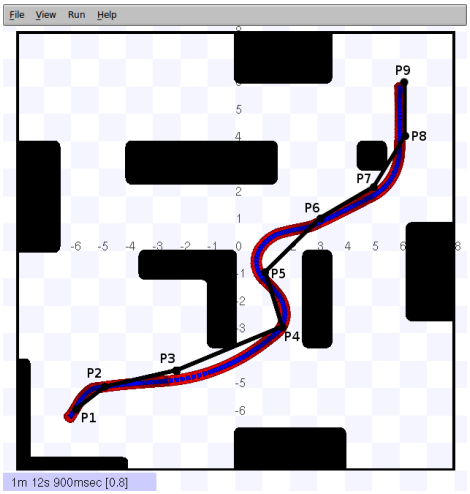

Fig. 11: Evolved Robotic Navigational Controller Application Demonstration

turn angle by the SDM subtask from the ideal values calculated using equation 2 and 4. The average and best SNN configuration error score minimises as the GA evolves leading to the minimum speed and direction output error.

4.4 Integration Subtask Design

The obstacle avoidance controller and speed and direction manager subtasks individually solve a functional aspect of the overall application. The Integration subtask (INT) combines the intermediate outputs from the OAC and the SDM

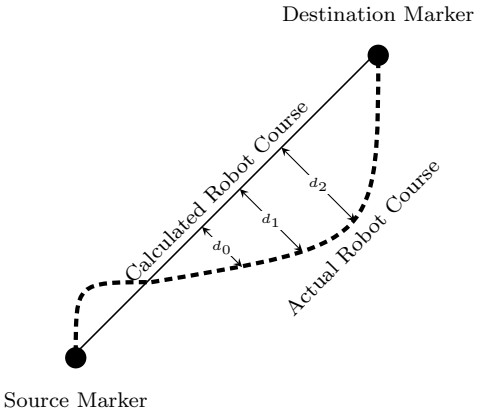

Fig. 12: Robot Course Deviation Calculation

$$F_{INT} = \frac{1}{\Sigma d_i} \tag{7}$$

Given:

Number of collisions = 0
SNN outputs are within the operating range ($0 \leq Speed \leq 5$ and $-3.14 \leq TurnAngle \leq 3.14$)
Robot does not spin indefinitely

application subtasks to generate the overall system output. The modular robotic navigational controller application (figure 6) is mapped onto three MNTs executing the OAC, SDM and INT subtasks in the robotic application evolution setup illustrated in figure 1. The application subtask MNTs are configured with the synaptic configuration obtained through SNN evolution as described in section 4.2 and 4.3. The INT subtask has six individual inputs (from the OAC and SDM application subtasks) and three outputs namely Speed, TurnAngle_0 and TurnAngle_1.

The application requires the robot to move from the source location and progress towards the destination avoiding the obstacles within the simulated two-dimensional environment. Figure 11 demonstrates the evolved robotic navigational controller application on the EMBRACE-FPGA platform. The pre-planned robot route (shown as the thinner line) progresses via the markers P_1, P_2, ... P_9. The accuracy of the integration subtask and the overall application is evaluated based on the deviation of the robot course from the marked route as depicted in figure 12, provided the robot completes the journey without colliding with the obstacles. The INT subtask is evolved using the fitness function illustrated in equation 7 (which is based on the accumulated course deviation error). Figure 13 illustrates the average and best robot course deviation score for the evolution of the INT subtask on the MNT. The actual path of the robot (controlled by the evolved hardware SNN) is shown as a thick line in figure 11.

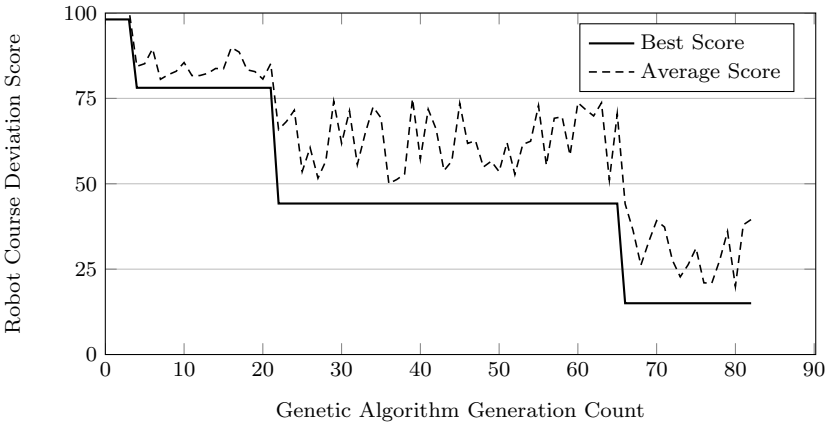

Fig. 13: Integration Subtask Evolution
(Note: The measured robot course deviation is capped at 100.)

4.5 Application Orthogonal Functionality Integration

Real-world applications often encounter scenarios where no unique solution can satisfy all the functional aspects of the application. These distinct functional aspects can have contradicting requirements for certain application scenarios which makes it impossible to have a correct solution for the whole application. Often a balanced solution satisfying all the functional aspects to a certain extent helps to solve the overall application and overcome the application scenario gracefully. One such scenario in the robotic navigational controller application is manoeuvring the robot around obstacle corners. The ideal solution for advancing the robot towards the next marker requires the robot to move towards the (obstructed) destination, whereas the ideal solution for avoiding the nearby obstacles requires the robot to steer away from the nearest obstacle.

The application prototyping technique presented in this paper comprises the modular application design and subtask evolution where each functional element of the application is evolved separately and the intermediate outputs are integrated to generate the system output. The integration subtask is trained to identify the input patterns for the contradicting application scenarios and produce a balanced output. The approach generates a balanced solution in various application scenarios (e.g. robot path from marker P_3 to P_4 and P_7 to P_8 in figure 11) and efficiently navigates the robot on the marked course. The proposed modular application prototyping technique offers effective integration of the orthogonal functions of the application.

4.6 Application Evolution Complexity Reduction

The individual SNN configuration in GA-based search comprises synaptic weight (5-bit signed value) and threshold potential (16-bit unsigned value) of all the neurons within the SNN. The binary coded SNN configuration has the search space of 2^n

Table 1: Robotic Navigational Controller Application SNN Configuration Comparion

Application Evolution	Modular				Non-Modular
	OAC	SDM	INT	Modular Total	
Number of neurons	19	8	9	36	36
Number of synapses	304	40	54	398	398
Bits in the GA gene	6384	328	414	7126	7126
GA search space	2^{6384}	2^{328}	2^{414}		2^{7126}
Total GA search space	$(2^{6384} + 2^{328} + 2^{414})$				2^{7126}

for n bits of configuration information. Increasing the SNN size (number of neurons and synapses) increases the GA search space and complexity by a factor of $O(2^n)$. The proposed modular SNN evolution technique is compared with the non-modular SNN evolution. Table 1 illustrates the SNN configuration comparison for the robotic navigational controller application for the modular and the non-modular evolution. The non-modular SNN evolution approach requires that all the three subtask SNNs are evolved concurrently, which results in a very large solution search space of 2^{7126}. The modular approach evolves each subtask separately resulting in a considerably smaller solution search space of $2^{6384} + 2^{616} + 2^{414}$. The modular SNN evolution results in smaller solution search space, mitigating the SNN evolution problem many folds and speeding-up the SNN evolution process.

5 Conclusions

Biologically-inspired hardware SNN architectures offer elegant solutions as low-power and scalable embedded computing platforms ideally suited for data/pattern classification, estimation and control applications. Various architectures comprising reconfigurable and highly interconnected arrays of neural network elements in hardware have been proposed to produce computationally powerful and cognitive signal processing units. One of the main design challenges for the realisation of practical hardware SNN systems is an efficient rapid application prototyping method.

This paper briefly reviewed the previously reported EMBRACE hardware SNN architecture comprising modular neural tiles interconnected using a hierarchical NoC communication infrastructure. The EMBRACE FPGA prototype employs GA-based SNN training which is effective for evolving small uni-functional applications on hardware SNN platforms, but the technique has a number of limitations including poor scalability and search space explosion for the evolution of complex SNN applications. The paper analysed the limitations of a GA-based SNN evolution in solving increasingly complex applications with orthogonal functional goals such as a robotic controller application with contradicting behavioural requirements in certain scenarios.

This paper presents a rapid application prototyping technique for hardware SNN architectures. The technique comprises modular application design and gradual GA-based evolution of subtasks for the orthogonal application sub-functions. The

modular application design process involves decomposing a complex application into orthogonal subtasks based on system input/output grouping and goals or functional aspects of the application. The intermediate outputs from subtasks are combined using an integration subtask that is evolved to identify the input patterns for contradicting application scenarios and produce a balanced output. The approach helps produce a dynamic system output that is capable of overcoming the application scenario gracefully. Structured layering of the application comprising multiple application subtasks feeding to higher level integration subtasks leads to an extensible application organisation where new functionality can be added in the form of application subtasks. Also, adding multiple application subtasks assigned for the same functionality but operating on different inputs (e.g. sonar and laser range finder inputs for robot proximity detection) adds the robustness to the application behaviour.

The GA-based evolution of large SNN structures for complex application requirements results in massive solution search space and exhibits poor scalability. The proposed modular evolution technique results in a smaller solution search space, mitigating the SNN evolution problem many folds and speeding-up the SNN evolution process. Real-world applications comprise many functional and behavioural aspects for which a solution can be built using a number of application modules solving distinct functional elements. The robotic navigational controller application presented in this paper demonstrates successful obstacle avoidance and route following characteristics even for the contradicting scenarios. The proposed technique offers a rapid, simple and effective application prototyping method for hardware SNN architectures.

Acknowledgements This research is supported by the International Centre for Graduate Education in Micro and Nano-Engineering (ICGEE), Irish Research Council for Science, Engineering and Technology (IRCSET) and Xilinx University Programme.

References

1. Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.
2. Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.
3. Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, August 2002.
4. Sander Bohte and JoostN Kok. Applications of spiking neural networks. *Information Processing Letters*, 95(6):519–520, 2005.
5. Martin Pearson, Anthony Pipe, Benjamin Mitchinson, Kevin Gurney, Chris Melhuish, Ian Gilhespy, and Mokhtar Nibouche. Implementing spiking neural networks for real-time signal-processing and control applications: a model-validated FPGA approach. *Neural Networks, IEEE Transactions on*, 18(5):1472–1487, 2007.
6. Jim Harkin, Fearghal Morgan, Liam McDaid, Steve Hall, Brian McGinley, and Seamus Cawley. A reconfigurable and biologically inspired paradigm for computation using network-on-chip and spiking neural networks. *Int. J. Reconfig. Comput.*, 2009:2:1–2:13, January 2009.
7. Seamus Cawley, Fearghal Morgan, Brian McGinley, Sandeep Pande, Liam McDaid, Snaider Carrillo, and Jim Harkin. Hardware spiking neural network prototyping and application. *Genetic Programming and Evolvable Machines*, 12:257–280, 2011.
8. Sandeep Pande, Fearghal Morgan, Seamus Cawley, Tom Bruintjes, Gerard Smit, Brian McGinley, Snaider Carrillo, Jim Harkin, and Liam McDaid. Modular neural tile architecture

for compact embedded hardware spiking neural network. *Neural Processing Letters*, pages 131–153, October 2013.
9. Sandeep Pande, Fearghal Morgan, Gerard Smit, Tom Bruintjes, Jochem Rutgers, Brian McGinley, Seamus Cawley, Jim Harkin, and Liam McDaid. Fixed latency on-chip interconnect for hardware spiking neural network architectures. *Parallel Computing*, 39:357 – 371, September 2013.
10. Sandeep Pande, Fearghal Morgan, Seamus Cawley, Brian McGinley, Jim Harkin, Snaider Carrillo, and Liam McDaid. Addressing the hardware resource requirements of Network-on-Chip based neural architectures. In *NCTA, International Conference on Neural Computation Theory and Applications, Paris, France*, October 2011.
11. Fearghal Morgan, Seamus Cawley, B McGinley, Sandeep Pande, LJ McDaid, Brendan Glackin, John Maher, and Jim Harkin. Exploring the evolution of noc-based spiking neural networks on FPGAs. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 300–303. IEEE, 2009.
12. Rodney Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.
13. Richard Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2:189–208, August 2008.
14. Snke Johannes, Bernardina M. Wieringa, Mike Matzke, and Thomas F. Mnte. Hierarchical visual stimuli: electrophysiological evidence for separate left hemispheric global and local processing mechanisms in humans. *Neuroscience Letters*, 210(2):111–114, May 1996.
15. David Van Essen, Charles Anderson, and Daniel Felleman. Information processing in the primate visual system: an integrated systems perspective. *Science*, 255(5043):419–423, January 1992.
16. Tom Binzegger, Rodney Douglas, and Kevan Martin. Stereotypical bouton clustering of individual neurons in cat primary visual cortex. *The Journal of Neuroscience*, 27(45):12242–12254, November 2007.
17. Bart Happel and Jacob Murre. Design and evolution of modular neural network architectures. *Neural networks*, 7(6):985–1004, 1994.
18. Gasser Auda and Mohamed Kamel. Modular neural networks: a survey. *International Journal of Neural Systems*, 9(02):129–151, 1999.
19. Eric Ronco and Peter Gawthrop. Modular neural networks: a state of the art. *Rapport Technique CSC95026 Center of System and Control University of Glasgow*, 1:1–22, 1995.
20. Daniel Osherson, Scott Weinstein, and Michael Stob. Modular learning. pages 369–377. MIT Press, Cambridge, MA, USA, 1993.
21. Sheng-uei Guan, Shanchun Li, and Syn Kiat Tan. Neural network task decomposition based on output partitioning. *Journal of the Institution of Engineers Singapore*, 44:78–89, 2004.
22. Bao-Liang Lu and Masami Ito. Task decomposition and module combination based on class relations: a modular neural network for pattern classification. *Neural Networks, IEEE Transactions on*, 10(5):1244–1256, sep 1999.
23. Jekanthan Thangavelautham and Gabriele Deleuterio. A neuroevolutionary approach to emergent task decomposition. In *Proc. of 8th Parallel Problem Solving from Nature*, pages 991–1000. Springer, 2004.
24. Vineet Khare, Xin Yao, Bernhard Sendhoff, Yaochu Jin, and Heiko Wersing. Co-evolutionary modular neural networks for automatic problem decomposition. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2691–2698. IEEE, sept 2005.
25. Jorge Santos, Luis Alexandre, and Joaquim Marques de Sá. Modular neural network task decomposition via entropic clustering. In *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, volume 1, pages 62–67. IEEE, oct. 2006.
26. Donald Olding Hebb. *The organization of behavior; a neuropsychological theory*. Psychology Press, 2005.
27. Geoffrey Hinton and Terrence Sejnowski. *Unsupervised learning: foundations of neural computation*. The MIT press, 1999.
28. Thomas Natschlaeger and Berthold Ruf. Online clustering with spiking neurons using temporal coding. *Progress in Neural Processing*, pages 33–42, 1998.
29. Sander Bohte, Han La Poutré, and Joost Kok. Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer rbf networks. *Neural Networks, IEEE Transactions on*, 13(2):426–435, 2002.

30. Florian Landis, Thomas Ott, and Ruedi Stoop. Hebbian self-organizing integrate-and-fire networks for data clustering. *Neural computation*, 22(1):273–288, 2010.
31. John J Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376(6535):33–36, 1995.
32. Wulfram Gerstner and J Leo van Hemmen. Associative memory in a network of spiking neurons. *Network: Computation in Neural Systems*, 3(2):139–164, 1992.
33. Masood Zamani, Alireza Sadeghian, and Sylvain Chartier. A bidirectional associative memory based on cortical spiking neurons using temporal coding. In *IJCNN*, pages 1–8, 2010.
34. Ethem Alpaydin. *Introduction to machine learning*. The MIT Press, 2010.
35. Sander Bohte, Joost Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1):17–37, 2002.
36. Robert Legenstein, Christian Naeger, and Wolfgang Maass. What can a neuron learn with spike-timing-dependent plasticity? *Neural Computation*, 17(11):2337–2382, 2005.
37. Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, 4(10):e1000180, 2008.
38. Dorit Baras and Ron Meir. Reinforcement learning, spike-time-dependent plasticity, and the bcm rule. *Neural Computation*, 19(8):2245–2279, 2007.
39. Michael Farries and Adrienne Fairhall. Reinforcement learning with modulated spike timing–dependent synaptic plasticity. *Journal of neurophysiology*, 98(6):3648–3665, 2007.
40. Razvan Florian. A reinforcement learning algorithm for spiking neural networks. In *Symbolic and Numeric Algorithms for Scientific Computing, 2005. SYNASC 2005. Seventh International Symposium on*. IEEE, 2005.
41. Razvan Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, 19(6):1468–1502, 2007.
42. Eugene Izhikevich. Solving the distal reward problem through linkage of stdp and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452, 2007.
43. Eleni Vasilaki, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS computational biology*, 5(12):e1000586, 2009.
44. Ezequiel Di Paolo. Spike-timing dependent plasticity for evolved robots. *Adaptive Behavior*, 10(3-4):243–263, 2002.
45. Hani Hagras, Anthony Pounds-Cornish, Martin Colley, Victor Callaghan, and Graham Clarke. Evolving spiking neural network controllers for autonomous robots. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4620–4626. IEEE, 2004.
46. Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
47. Dmitri Vainbrand and Ran Ginosar. Scalable network-on-chip architecture for configurable neural networks. *Microprocessors and Microsystems*, 35(2):152–166, 2011.